Group Construction for Airline Cabin Crew: Comparing Constraint Programming with Branch and Price

Jesper Hansen[1] and Tomas Lidén[2]

Carmen Systems
[1] Købmagergade 53, DK-1150 København K, Denmark
[2] Maria Bangata 6, SE-118 63 Stockholm, Sweden
{jesper.hansen, tomas.liden}@carmensystems.com

Abstract. Producing work schedules for airline crew normally results in individually different schedules. Some airlines do however want to give the same schedule to groups of people. The construction of such groups must respect certain rules, provide a good matching of certain factors and fit well into the normal process of producing anonymous trips starting and ending at the home base and assigning these to the crew. In this paper we present an application, implemented and delivered to a large European airline, which addresses these needs. The problem is challenging to solve for certain cases. Hence two different approaches have been applied, one using constraint programming and the other using column generation. These two methods are described and compared – along with computational results.

1 Introduction

Constructing work schedules for airline crew is typically divided into a crew pairing problem and a crew rostering problem. In the pairing problem anonymous pairings, or trips starting and ending at the home base, are constructed from the flight legs such that the crew need of each flight is covered. Following the pairing construction, the pairings are assigned to individual crew together with other activities such as ground duties, reserve duties and off-duty blocks, to form *rosters*. For more information on crew pairing and rostering we refer to the surveys of Andersson et. al. [1] and Kohl and Karisch [7].

Some airlines want to give the same work schedule/roster to a group of cabin crew members (purser and cabin attendants). This is to ease planning, increase the robustness of the schedule and for social reasons. Constructing such groups can be done before the rostering step. Then, a representative person from each group (normally the purser) is assigned a roster by the crew rostering system, which is then copied to the rest of the group members. If the group has been poorly formed, it will not be possible to copy all pairings. For example, if one crew member has a pre-assigned duty (e.g. course), a pairing touching that day cannot be assigned. Such "drop-out" pairings must be resolved manually.

R. Barták and M. Milano (Eds.): CPAIOR 2005, LNCS 3524, pp. 228 – 242, 2005.

The construction of crew groups should therefore be solved so that the number of problems to handle in successive steps is minimized (both for the current scheduling period and future ones). Thus we want to achieve "homogenous" groups.

An application solving the crew-grouping problem has been developed and delivered to the Spanish airline Iberia as an addition to the Carmen Crew Rostering system. It has been used in production since spring 2002. Each problem instance solved includes 300-1200 crew and results in 30-200 groups of 3-13 persons each.

There are several approaches to formulating and solving this problem. The first delivered version used pure Constraint Programming (CP) [2]. During spring 2004 a second version was delivered based on Column Generation (CG) where the generation makes use of CP techniques (enumeration with some simple look-ahead domain reduction). The idea of combining CG and CP is not new (see e.g. Junker et. al. [6]) and many papers have been dedicated to comparing Integer Programming and CP methods for various applications. Grönkvist [4, 5] for instance, compares the methods on the Aircraft Scheduling Problem.

The initial reasons for choosing CP were the uncertainty of the problem formulation and constraints involved. Further the initial descriptions indicated a highly constrained problem including non-linear constraints and objectives. The need for modeling flexibility and a restricted budget were also important factors.

During the elaboration and development of the CP version the problem proved to be less constrained than anticipated and turned out to be more of an optimization problem than a feasibility problem. The idea of a mathematical programming formulation started to evolve and was investigated in a master's thesis work [3]. The results were good, however not all issues were considered. Hence it was decided to develop a CG version. The normal cases (January-November) to be introduced in the following were easily solved, but the "December" problem for the medium and large fleets proved to be surprisingly hard to solve. Thus the development of several advanced features was needed, such as stabilized column generation [8], and branch and price using constraint branching [9]. All these features along with a careful tuning of parameters have been necessary in order to achieve the goal of finding better solutions within similar or shorter execution times.

After giving a problem description, the pure CP model is described followed by the CG model. In sections 4 and 5 the solution procedures are outlined. In section 6 we describe the special considerations for the month of December. Computational results are presented in section 7 along with discussions on the suitability of the different methods, properties of the problem and alternative approaches. We end with a conclusion in section 8.

2 Problem Description

The problem is basically to assign crew members to groups such that the best possible matching is achieved minimizing the exceptions to handle when assigning schedules to the groups. The number of crew per group depends on the aircraft type. The cost factors in the matching problem are:

- **Preferred days off:** Match crew with the same or overlapping wishes for days off. The matching is done towards the purser if she has any preferred days off, otherwise towards the crew with maximum number of preferred days off.
- **Pre-assignments:** Match crew with the same or overlapping pre-assignments (courses etc). This matching is also done towards the purser if she has any pre-assignments, otherwise towards the crew with the most pre-assignments.
- **Historic values:** Match crew with excess or deficit for selected fairness values (e.g. flights to certain destinations, overtime etc).
- **Work reduction:** Match crew with the same work reduction together. Specifically it should be avoided to mix fulltime and reduction crew in the same group.

For the month of December there are special considerations described in section 6. In the normal problem (January - November), the minimum number of groups shall be constructed. The total cost for the constructed groups, which we define later, shall be minimized.

A number of rules are considered:

- **Buddy Bids**. The crew members give buddy bids saying that two or more people want to fly together. These buddy bids are "closed", meaning that one person can only participate in one bid. The buddy bids must be respected. Crew in buddy bids must be placed in a group while other crew can supplement groups if necessary.
- **Pursers**. There must be exactly one purser in each group.
- **Incompatibles**. There can also be crew forbidden combinations, which must be avoided.
- **Granted Days Off**. The number of granted days off is limited and certain patterns of granted days off are not allowed: Before and after a sequence of two or more consecutive granted days off there must be a stretch of at least 3 working days. Allowed patterns where × is a granted day off are:

  ×000×× ××000×

  Not allowed patterns are:

  ×00×× ×0×× ××00× ××0×
- **Inexperienced/transitions**. Limits can be set on the number of inexperienced crew per group, or crew in a transition phase from one type of aircraft to another.
- **Work reduction**. Cabin attendants must have less or the same work reduction as the purser.

3 Model Formulation

For modeling purposes we consider all crew to be in a so-called *subgroup*. A subgroup can consist either of all crew in a buddy bid or a single crew in no buddy bid. A crew member belongs to only one subgroup. Further since a group must have exactly one purser, each subgroup with a purser is associated with exactly one group. Let S be the set of subgroups. Further S is partitioned into two parts: S^B containing subgroups that must be in a group (buddies) and S^C containing subgroups that do not necessarily have to be in a group (single crew). Let F denote the set of fairness factors.

3.1 The CP Model

We first present the normal model and then in section 6.1 present the additions necessary for handling the special considerations in December.

Each subgroup *i* has the following basic properties, calculated from its set of crew:

N_i	Number of crew in the subgroup.
D_i	A set of preferred days off.
D_i^G	A set of granted days off, which is a subset of D_i.
P_i	A set of pre-assigned days.
H_{fi}^L, H_{fi}^H	Lowest and highest historic value (integer) for fairness factor $f \in F$.
$N_{fi}^-, N_{fi}^0, N_{fi}^+$	Number of crew with negative, zero and positive historic value for fairness factor $f \in F$.
N_i^I, N_i^T	Number of inexperienced crew and crew in transition phase
R_i	Work reduction factor (takes values 5, 3, 2, 1 or 0, where 5 is half-time and 0 is full-time)

Let S_j denote the set of subgroups for group *j*, with the initial domain *S*. Thus the following properties can be calculated for the group:

$$n_j = \sum_{i \in S_j} N_i$$

$$d_j = \bigcup_{i \in S_j} D_i \qquad d_j^G = \bigcup_{i \in S_j} D_i^G \qquad p_j = \bigcup_{i \in S_j} P_i$$

$$h_{jf}^L = \min_{i \in S_j} \left(H_{fi}^L \right) \qquad h_{jf}^H = \max_{i \in S_j} \left(H_{fi}^H \right)$$

$$n_{jf}^- = \sum_{i \in S_j} N_{fi}^- \qquad n_{jf}^0 = \sum_{i \in S_j} N_{fi}^0 \qquad n_{jf}^+ = \sum_{i \in S_j} N_{fi}^+$$

$$n_j^I = \sum_{i \in S_j} N_i^I \qquad n_j^T = \sum_{i \in S_j} N_i^T$$

$$r_j^L = \min_{i \in S_j} \left(R_i \right) \qquad r_j^H = \max_{i \in S_j} \left(R_i \right)$$

Let N_j, D_j, etc denote the properties for the purser subgroup associated with group *j*. Then the cost c_j for group *j* is calculated, using weight factors W^D, W^P, W^{HS}, W^{HR}, W^R and cost factors as follows:

$$c_j = W^D c_j^D + W^P c_j^P + \sum_{f \in F} \left(W_f^{HS} c_{jf}^{HS} + W_f^{HR} c_{jf}^{HR} \right) + W^R c_j^R \tag{1}$$

– **Days Off**. Limit the number of non-matching days off compared to crew with max number (use purser as max if she has any days off).

$$c_j^D = \begin{cases} \left| d_j \right| - \left| D_j \right| & \text{if } \left| D_j \right| > 0 \\ \left| d_j \right| - \max_{i \in S_j} \left| D_i \right| & \text{if } \left| D_j \right| = 0 \end{cases} \tag{2}$$

– **Pre-Assignments**. Same as for days off.

$$c_j^P = \begin{cases} \left|p_j\right| - \left|P_j\right| & \text{if } \left|P_j\right| > 0 \\ \left|p_j\right| - \max_{i \in S_j} \left|P_i\right| & \text{if } \left|P_j\right| = 0 \end{cases} \tag{3}$$

– **Historic Values – Sign**. Try to get all crew on excess or deficit.

$$c_{jf}^{HS} = \min\left(n_{jf}^-, n_{jf}^+\right) \tag{4}$$

– **Historic Values – Range**. Minimize the span in historic values.

$$c_{jf}^{HR} = h_{jf}^H - h_{jf}^L \tag{5}$$

– **Reduction**. 1) Don't mix full-time and reduction. 2) Avoid spread in reduction.

$$c_j^R = \begin{cases} 3 + r_j^H - r_j^L & \text{if } r_j^H > 0 \text{ and } r_j^L = 0 \\ r_j^H - r_j^L & \text{if } r_j^H = r_j^L = 0 \text{ or } r_j^L > 0 \end{cases} \tag{6}$$

The first row in (6) is used when both full-time crew and crew with reduction are placed in the same group. Since this is considered especially bad, an extra penalty of 3 is added. The other row is used when all crew are full-time or have reduction.

The formulation of these costs has been designed in close collaboration with the customer and reflects what they consider as a desirable group construction. However, this formulation has several numerical problems, since factors (2) and (3) are non-additive (the cost does not increase monotonically when adding subgroups). This makes it difficult to calculate a good lower bound for the cost during the construction of a group and hence the possibility of pruning the search tree. The applied lower bound calculation for days off is given below.

$$\left\lfloor c_j^D \right\rfloor = \begin{cases} \left|d_j\right| - \left|D_j\right| & \text{if } \left|D_j\right| > 0 \\ 0 & \text{if } \left|D_j\right| = 0 \end{cases} \tag{2.1}$$

Let S_k^I denote the set of subgroups that should not be put in the same subgroup according to incompatibility k. Then the following constraints must hold for each group j:

– **Incompatibilities**

$$\left|S_j \cap S_k^I\right| \le 1 \qquad \forall k \tag{7}$$

– **Limit number of inexperienced crew and crew in Transition**

$$n_j^I \le L^I,\ n_j^T \le L^T,\ n_j^{I+T} \le L^{I+T} \tag{8}$$

– **Limit number of granted days off**

$$d_j^G \le L^G \tag{9}$$

– **No illegal patterns for granted days off**

$$NoIllegalPattern\left(d_j^G\right) \tag{10}$$

– **Reduction**. Same or less work reduction as the purser. Let $S^{CR} \subseteq S^C$ denote the set of single crew with reduction. Single crew with reduction cannot join a purser without reduction.

$$\begin{cases} r_j^H \le R_j & \text{if } R_j \neq 0 \\ S_j \bigcap S^{CR} \equiv \varnothing & \text{if } R_j = 0 \end{cases} \tag{11}$$

Of the above constraints, (9) has the most limiting effect.

During the construction of a group the above constraints will reduce the domain of S_j. This domain reduction is however rather weak, mainly due to the non-additive property of factors (2) and (3) – these values will therefore be bounded late in the construction. Moreover, since all constraints apply per group they will not propagate any domain reduction to other groups.

When solving a pure CP version of the problem, we also need the following global constraints:

– **Each subgroup is used only once**

$$\bigcap_j S_j \equiv \varnothing \tag{12}$$

– **All crew in buddy bids must be assigned to a group**

$$S^B \subseteq \bigcup_j S_j \tag{13}$$

This CP model does not take advantage of the fact that several subgroups may have the same characteristics that permits us to strengthen (12) and (13).

3.2 The CG Model

Let G be the set of all feasible groups that can be constructed from the subgroups. Now we can formulate the problem as the following Set Partitioning/Set Packing problem:

$$\begin{aligned} &\min \sum_{j \in G} \left(M + c_j\right) x_j \\ &s.t. \\ &\sum_{j \in G} a_{ij} x_j = b_i, \quad \forall i \in S^B \\ &\sum_{j \in G} a_{ij} x_j \le b_i, \quad \forall i \in S^C \\ &x_j \in \{0,1\}, \quad j \in G \end{aligned} \tag{14}$$

where c_j is the cost of group j and M is a large negative number if we are maximizing the number of groups, large positive if we are minimizing and 0 if we are only

focusing on costs. Further, a_{ij} is 1 if subgroup i is in group j, and 0 otherwise. When several subgroups have similar characteristics it is possible to aggregate the model so that the b_i's represent the number of similar subgroups of type i. Each subgroup type can then participate in the same group more than once and some subgroups can be used more than once resulting in the following limitations:

$$0 \le a_{ij} \le b_i \text{ and } 0 \le x_j \le \left\lfloor \min_{i \in S} \left(b_i / a_{ij} \right) \right\rfloor \tag{15}$$

The size of the set G is potentially exponential in the number of subgroups. The model is hence solved for a restricted number of groups where the values of the dual variables of the LP-relaxed model are used for finding new groups that correspond to columns with negative reduced cost. When no group exists with negative reduced cost the LP-relaxation is optimal. The problem of finding these groups is very similar to the problem formulated in section 3.1. Here we are searching for one feasible group at a time resulting in a new column in the model. The optimal solution to the LP-relaxation is however very unlikely to be integer. To get optimal integer solutions, the column generation needs to be embedded in a Branch & Bound procedure where columns can be generated in every node of the Branch & Bound tree. This procedure is referred to as Branch & Price.

4 Solving the CP Model

To solve the pure CP model, a customized constraint $NoIllegalPattern\left(d_j^G\right)$ was implemented. The filtering algorithm applies the following propagations whenever the domain of d_j^G changes.

Remove impossible values at start or end of a sequence (0 means not in domain, ? means possible in domain, × means required in domain. Affected values are underlined.)	0?×× → 0<u>0</u>×× ??0×× → <u>000</u>×× ××?0 → ××<u>0</u>0 ××0?? → ××<u>000</u>
Add required values at start or end of a sequence	×?×× → ×<u>×</u>×× ×??×× → ×<u>××</u>×× ××?× → ××<u>×</u>× ××??× → ××<u>××</u>×

We then apply the following three-step approach:

1. Initial solution plus refinement. The search strategy selects a group (variable S_j) and then finds a suitable subgroup (value) to assign to that group. This continues in a depth-first search. Having found an initial solution, the refinement process adds additional constraints that improve the quality of the solution by setting limitations on the max values of c_j^D (days off), c_j^P (pre-assignments), c_{fj}^{HS} (historic values, sign) and then resolving the problem. This approach has been used to reduce the size of the problem and increase the domain reduction.
2. Global search for cheaper solutions. The solution found in step 1 is further improved by adding a constraint that forces the total cost to decrease for each new

solution and then search for better solutions. The same search strategy is used as in step 1. When the cost decreases slowly, some additional constraints are added that forces very expensive groups to get a lower cost (and hence be modified). Due to the very large search tree this step cannot be completed within reasonable time. Thus early decisions made in the search strategy are not likely to be altered.

3. Local search for cheaper solution. In this final step, groups are picked in pairs from the current solution and all possible solutions for these two groups with their used subgroups are calculated to see if a cheaper solution can be found. All pairs of groups are treated in this way. This step is repeated until no better solution can be found.

It should be noted that the pure CP model does not assume minimizing/maximizing the number of groups. Instead this is built into the search heuristic by first assigning buddy bids (large before small) and then filling up the groups to the desired size. If the minimum number of groups is wanted, no more groups are constructed. If the maximum number of groups is wanted, the algorithm continues constructing groups until the available groups or subgroups are exhausted. This method does not guarantee to produce the minimum possible number of groups, but will come reasonably close.

The order in which groups and subgroups are picked and assigned is crucial, since that will set the quality of the solution after step 1 and 2. The ordering is a trade-off between greediness (to find high-quality/low-cost solutions) and feasibility (to actually find solutions). Since the domain reduction is rather poor (see the end of section 3.1) we get a very large search tree. Therefore a very greedy approach can easily result in not finding feasible solutions within a reasonable computation time.

The search heuristic gives special consideration to granted days off, since that is crucial for finding feasible solutions (due to constraint (9) and (10)). Apart from that, it looks at the domain of the cost (c_j) and the lower bound of the additional cost when adding different subgroups. If a good solution has been found after step 2, the local search will only make minor adjustments, but if the solution has poor quality after step 2, the last step will repeat many times and can therefore be very time-consuming (although it is surprisingly effective in decreasing the cost).

Several drawbacks can be noted regarding the local search in step 3: It only uses subgroups that are already in the solution (not unused subgroups); it cannot achieve modifications that include swapping subgroups between three or more groups; it cannot escape from local minima.

5 Solving the CG Model

To solve the CG model we apply the following three-step approach:

1. Set-up an initial problem matrix with a) columns from a simple initial construction heuristic, which ensures the existence of at least one IP-feasible solution and gives an upper bound, b) columns from an initial generation.
2. Solve the LP-relaxation and perform pricing (column generation) to generate better columns based on the dual values until no better columns can be constructed.
3. Perform branch & price until no better solution can be found or the time limit is reached. When the time limit is reached a final IP is solved consisting of all obtained columns and no variables fixed.

5.1 The Pricing Problem

A constraint handling procedure is implemented for testing legality of a group. With this procedure a subgroup infeasibility table is set-up: For each subgroup we have a set of subgroups for which the subgroup is pair-wise infeasible. During the construction of groups this infeasibility table is used to reduce the domain of the problem. The constraint handling is used to verify the legality of each group before and after adding a subgroup to it.
The generation of columns is done by considering one group at a time and assigning subgroups from a list of feasible subgroups in a depth first search procedure. The list of subgroups to consider is randomized (to increase search coverage) and sorted according to dual value (positive before zero before negative). During the assignment of subgroups two pruning techniques are applied:

- **Cost cuts:** When assigning a subgroup with dual value ≤ 0 we know that the sum of duals cannot increase (due to the sorting applied). Then we calculate a lower bound for the group cost, which is also a lower bound on the reduced cost. When the lower bound becomes positive, we stop exploring the current node and backtrack.
- **Pattern cuts:** The constraint $NoIllegalPattern\left(d_j^G\right)$ can be used to check whether any specific values must be added to d_j^G in order to maintain legality. For example: if $d_j^G = \{5,8,9,10\}$ we know that 6 and 7 must be added – otherwise the constraint will not hold. In such situations we loop over the remaining subgroups to see if these needed additions can be found. If not, there is no need to explore the current node any further and we backtrack. During the looping over remaining subgroups we can also remove all subgroups that are infeasible with the currently assigned ones according to the infeasibility table and any subgroups that would result in breaking any constraints. If this results in an empty list of remaining subgroups we backtrack.

Apart from the above no special domain reductions are used. We have investigated the use of no-goods, forward-checking etc, but the computational effort is not worthwhile. Solving the CP model in a CP framework as in [3] could also have been used, but the domain reduction did not seem to be sufficient for introducing the extra overhead.

Only columns that will improve the LP solution (according to the reduced cost) will be used. In addition the number of backtracks in each node is limited and the generation reverts to the top node as soon as a valid column has been found. Thus we get a form of backtrack-bounded search that will get a broad coverage of the search tree instead of going deep into the search tree. This will improve the quality of the dual information. A limited time of a few seconds is spent on constructing columns for each group. There is hence no guarantee of finding a column with negative reduced cost even if such exists and we cannot prove that no column exists with negative reduced cost. To speed up the generation, we stop generating for a group, if no columns were generated for the group in the previous iteration. When no groups remain, we restart generating for all groups again. This greatly reduces the wasted time of unsuccessful search for columns.

The above handling relies heavily on the dual values from the previous LP solution. For some problem instances the dual values can fluctuate substantially, resulting in larger uncertainty and more iterations needed to converge. To remedy such cases, stabilized column generation due to du Merle et. al. [8] has been implemented. An alternative to this approach is to use an interior point or subgradient solver resulting in more well-behaved duals.

As can be seen from the above, the column generation does not have any problem specific considerations.

5.2 Branch and Price

Branching is performed if no columns were generated in the last pricing loop or the reduced cost of the best column is above a certain threshold. Note that this only applies if the search for columns is done for all groups.

Constraint branching according to Ryan and Foster [9] is applied for pairs of subgroups that have the partitioning constraint =1. The interpretation in this context is that either two subgroups must be in the same group or they must not be in the same group. When branching on the packing constraints ≤1, at least one of the constraints must be a partitioning constraint. Otherwise we cannot force them to be together since only one of them has to be used.

When finding the best branching candidate (a pair of subgroups) and its fractional sum *f*, the following branching scheme is applied:

Find *f* closest to 1. If *f* is larger than some threshold value (parameter setting) then apply the 1-branch, which bans columns with only one of the two subgroups, and continue branching. The 0-branch is not investigated upon backtrack.

If *f* is smaller than the threshold value then pricing is performed both in the 1-branch and the 0-branch. Continue branching, first in the branch that has the best LP objective and in the other when backtracking. When applying the 0-branch columns are banned which include both subgroups.

When no more branching candidates are found, we search for an improving IP solution and backtrack.

To fathom a node in the tree the lower bound of the node must be equal or larger than the best upper bound found so far. The LP bound is however only a valid lower bound if no column with a negative reduced cost exists. To establish this fact would require a complete enumeration of the pricing problem, which is generally not possible. Since we have no ambition of solving the problem to optimality by completely enumerating the branching tree, we can cut off a branch, which could contain an improving solution by assuming that the LP objective is a valid lower bound. So, if the LP objective in a node is worse than the best found upper bound, that node is fathomed and we backtrack.

The branching tree is only explored to a limited depth. If the max depth or time limit is reached we call the IP solver to search for a better IP solution and backtrack to the top of the tree. At the end of the search we finally call the IP solver to search for an improving solution on all generated columns.

After a branching decision the pricing procedure must ensure that columns are generated which follows earlier made branching decisions. For a specific group and corresponding purser subgroup we can identify branching decisions, which force a set

of other subgroups to be together with this subgroup. These can be added a priori. Given the remaining branching decisions a number of clusters of subgroups can be identified, which are forced to be together. During the pricing search procedure, clusters are added before individual subgroups not in clusters. When adding both clusters and individual subgroups, we check that this does not violate any branching decision that two subgroups must not be together. This procedure guarantees that all branching decisions are fulfilled.

The constraint branching scheme does not apply for constraints where $b_i>1$, $i \in S$. In that case one could use the general branching scheme of Vanderbeck and Wolsey [10]. Both the branching and the pricing procedure would however be even more complex and has hence not been implemented.

6 Special Considerations in December

We have in this section collected all problem specifics for the month of December. In December the following changes apply compared to the normal case:

- The number of groups shall be maximized.
- There is special consideration to pre-assignments – any pre-assignments on New Years Day (called *day32*) must be very well matched.
- Due to the strong focus on day32, buddy bids containing both persons with and without day32 will be split in two parts. If possible, perfect matching of day32 is wanted. If not, it is preferred that split bids are rejoined.

6.1 Modeling the December Problem

First we introduce an additional basic property per subgroup

B_i Other subgroup (the one without day32) from split buddy bid

and then calculate the following parameters per group: $n_j^{32} = \sum_{\substack{i \in S_j \\ 32 \in P_i}} N_i$, $b_j = \bigcup_{i \in S_j} B_i$

The cost function is modified so that an additional element is added to c_j:

$$c_j = [\text{as given in (1)}] + W^{P32} c_j^{P32} \tag{16}$$

To avoid combining crew with and without day32, and achieve rejoining of buddies from split bids, the cost factor and lower bound for pre-assigned day32 are calculated as follows:

$$c_j^{P32} = \begin{cases} 0 & \text{if } n_j^{32} = 0 \\ n_j^{0,not\_rejoined} + \dfrac{n_j^{0,rejoined}}{n_j} & \text{if } n_j^{32} > 0 \end{cases} \tag{17}$$

$$\text{where} \quad n_j^{0,not\_rejoined} = n_j - n_j^{32} - n_j^{0,rejoined}$$

$$n_j^{0,rejoined} = \sum_{i \in (S_j \cap b_j)} N_i$$

$$\left\lfloor c_j^{P32} \right\rfloor = \begin{cases} 0 & \text{if } n_j^{32} = 0 \\ \dfrac{n_j - n_j^{32}}{n_j} & \text{if } n_j^{32} > 0 \end{cases} \tag{17.1}$$

6.2 Solving the CG Model in December

The search heuristic is modified for this problem, taking the day32 factor into consideration. The crucial thing is to decide whether to go for rejoining buddy bids or not, however the demand for always obtaining a feasible solution restricts the amount of greediness.

6.3 The Pricing Problem for December

A few tricks are applied in the pricing phase to make it work better for the December problem. The sorting of subgroups during generation is slightly refined, and as soon as a possible rejoining of buddy bids is found, that rejoining is tried as well.

7 Computational Results

The application has been implemented in C++. The CP version uses ILOG Solver 5.2 as the constraint solver library, while the CG version uses XPress-MP 2003F as the optimization engine (with Coin/Osi as the programming API).

The system has been run for all fleet types used at Iberia – the below table show some basic data.

Fleet name	Group size	No. of crew	No. of subgroups	Produced no. of groups
MD87	3	~350-500	~180-230	~100-150
A320	4	~700-900	~310-450	~150-200
B757	5	~300-400	~150-200	~40-80
A340	10	~1000-1200	~350-420	~90-110
B747	13	~400-500	~40-200	~30-40

In the following table computational results are presented for a series of instances. For the CG version three execution times are reported: time for first solution, best solution and total execution time (a time limit of 60 minutes for the normal instances and 180 minutes for the large December instances (A340 and B747) has been used).

Some comments concerning the results:

- CG is generally able to find better solutions than CP. It minimizes the number of groups if possible and it yields better matching of day32 in December.
- Instances for small groups (group size <= 5) are generally solved faster with CG.
- Instances for large groups (group size > 5) require substantial amount of time and are generally slower with CG.

Num groups Total cost Σc_j Execution time (min)[1]	CP	CG	CG-CP
MD87	104 985,278 3.7	104 870,356 0.1/0.1/0.1	-11.7 % -3.6/-3.6/-3.6
A320	150 1,966,488 14.2	150 1,643,148 0.2/0.2/0.2	-16.4% -14.0/-14.0/-14.0
B757	38 437,446 1.5	35 425,913 0.1/4.3/4.3	-3 groups -2.6% -1.4/+2.8/+2.8
A340	82 521,000 9.1	73 502,000 13.6/61.1/61.1	-9 groups -3.6% +4.5/+52.0/+52.0
B747	30 476,335 2.2	26 445,554 0.8/30.1/30.1	-4 groups -6.5% -1.4/+27.9/+27.9
MD87, December	120 148,966 3.9	120 113,985 0.2/0.2/30.1	-23.5% -3.7/-3.7/+26.2
A320, December	198 3,309,710 32.0	198 3,042,610 2.4/55.0/55.0	-8.1% -29.6/+23.0/+23.0
B757, December	68 249,204 2.4	68 173,245 1.1/15.2/15.3	-30.5% -1.3/+12.8/+12.9
A340, December	110 560,935 67.6	110 470,909 51.0/70.5/181.1	-16.0% -16.6/+2.9/+113.5
B747, December	31 285,880 1.3	31 241,046 1.3/129.9/129.9	-15.7% -0.0/+129/+129
B747, December with no crew excess	39 909,892 4.6	39 1,076,933 76.9/181.5/181.5	+18.4 +72.3/+177/+177

- For cases where there is no or very little excess of crew for group construction (last example), CP finds better solutions more quickly than CG.
- A340 is the hardest since it both has a large number of subgroups and has a large group size, which gives a large combinatorial complexity.

[1] The results have been produced on the following platforms: SunOS 2.7 on a SparcServer 10000 with 16 UltraSparcII, 250MHz each, (CP) and Linux on a PentiumIII, 1300 MHz. The SPECint2000 figures are about 130 for the Sparc and 600 for the Pentium. Thus the times in the CP column are a result of dividing the actual CP times by 4.6.

- To achieve good matching for day32 for December, the tuning of parameters is crucial. The application has parameters for setting tolerance, min/max number of columns to generate per group and iteration, branching depth, threshold, max number of backtracks per generation node etc.

Another difference is that the CP version will improve the solution over the execution time and stop when it fails to do so. The CG version however will find a very good solution early, which can be slightly improved, but it will take very long time to do so – see Fig.. Therefore the user has the possibility to stop the execution whenever an acceptable solution has been found.

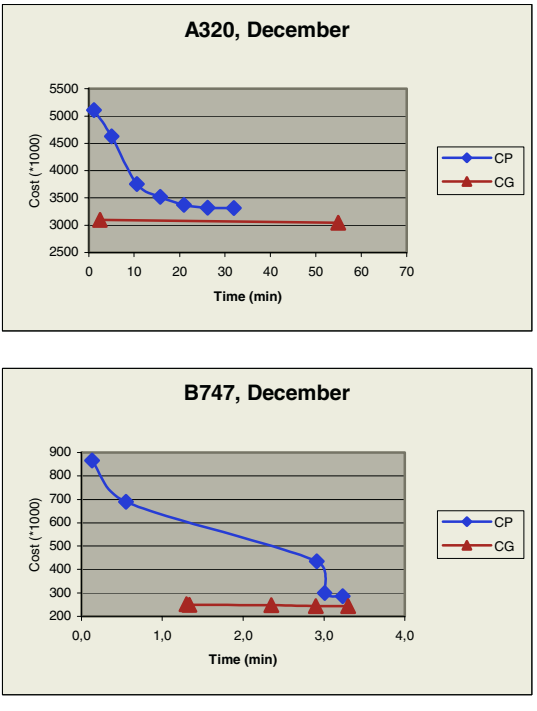

Fig. 1. Solution progress

8 Conclusion

First it should be noted that the numerical properties of the cost factors and constraints are a major cause for the difficulties of this problem. Reformulating some of these factors would probably be beneficial but such a redesign together with the customer was not possible to perform. Also, it is not obvious that alternative formulations exist that fulfills the needs of the customer.

There are some additional properties that make this problem hard to solve even with decomposition: It is a Set Partitioning/Packing Problem where there is no natural ordering (like connection time in crew pairing/rostering and distance in vehicle

routing). Instead there are multiple factors that should be matched well, and these factors have no correlation between subgroups.

When comparing CP and CG for this problem it is not easily said that one is better than the other. The expressiveness and ease of modeling was beneficial for the development of the CP version. The CG version does however perform better – after the inclusion of some rather sophisticated techniques. Further enhancements could of course be applied to the CP version. For example, it could consider subgroups with similar properties (which reduce both domains and symmetries), taking advantage of the mirrored formulation/channeling constraints (letting the subgroup have a variable for which group to belong to) and using a meta-heuristic like Tabu search instead of the simple local search adopted.

On the other hand the CG formulation is more general, since it only has one problem specific consideration (day32) built into the solving methods while the CP search heuristic is much more tailored to the specific problem properties. Thus the CG version should be more stable to future additions and changes, which was also a reason for choosing that method. And there are of course further enhancements that could be considered to the CG version.

References

1. Andersson, E., Housos, E., Kohl, N., Wedelin, D.: *Crew pairing optimization,* in: OR in airline industry, Eds. Yu, G., Kluwer Acad. Publ., 1997.
2. Carmen Consulting: *Carmen Crew Grouping Module*, (Product sheet), http://www.carmenconsulting.com, 2002.
3. A. Ekenbäck: *Optimal Crew Groups – A column generation heuristic for a combinatorial optimization problem*, Royal Institute of Technology, Sweden, (Master's Thesis), 2002.
4. M. Grönkvist: *Using Constraint Propagation to Accelerate Column Generation in Aircraft Scheduling*, In proceedings of CP-AI-OR'03, 2003.
5. M. Grönkvist: *A Constraint Programming Model for Tail Assignment*, In proceedings of CP-AI-OR'04, 2004.
6. U. Junker, S.E. Karisch, N. Kohl, B. Vaaben, T. Fahle and M. Sellmann: *A Framework for Constraint Based Column Generation,* In proceedings of CP'99, 1999.
7. N. Kohl and S. E. Karisch: Airline Crew Rostering: Problem types, modeling, and optimization, Annals of Operations Research 127, 223—257, 2004.
8. O. du Merle, D. Villeneuve, J. Desrosiers and P. Hansen: *Stabilized Column Generation*, Discrete Mathematics 194, 229—237, 1999.
9. D. M. Ryan and B. A. Foster: *An integer programming approach to scheduling*, In A. Wren, editor, Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling, 269—280, North-Holland, 1981.
10. F. Vanderbeck and L. A. Wolsey: *An exact algorithm for IP column generation*, Operations Research Letters, 19, 1996.